University of Pittsburgh

Optimization by Parametric Sweep

Week 11

MEMS 1140—Introduction to Programming in Mechanical Engineering

Learning Objectives (L.O.)

At the end of this lecture, you should understand/be able to:

- ❑ What optimization means in engineering;
- ❑ What it means to optimize by parametric sweep;
- ❑ Apply parametric sweep optimization to bridge design.

Table of Contents (ToC)

1 – What optimization means

The word “optimization” is used all the time in engineering.

These problems can be very simple or *extremely* complex.

In general, “to optimize” means to maximize or minimize one or more performance metric of a system (e.g. a part’s weight)

There is usually at least one constraint. In engineering, we consider things like safety, cost, material usage, etc.

1 – Optimization in MEMS

This lecture is intended as an introduction to the concept.

Here are just a few of the ME courses that incorporate optimization in some way:

- MEMS 0051 (Introduction to Thermodynamics);
- ENGR 0145 (Statics & Mechanics of Materials 2);
- MEMS 1141/2 (Data Science for Engineers);
- MEMS 1014 (Dynamic Systems);
- MEMS 1065 (Thermal Systems Design).

1 – Optimization Complexity

There are *many* different optimization methods. Some are designed for specific cases and have both pros and cons.

This lecture and next will only cover two methods.

Parametric sweep is robust - it accurately returns the global extremum within the domain - but it can be very slow.

Gradient descent is faster, but it is only guaranteed to return the global extremum if the function is known to be unimodal.

2 – Parameterized Problems

All problems have at least one variable that you can adjust as the designer to vary solution performance. This a parameter.

For example: You could laser cut from either cardboard, wood, or acrylic, or even 3D print parts for your MEMS 0024 rovers.

A parametric sweep means to test a domain of values for each parameter and evaluate the solution performance.

The optimized combination of parameters performs best.

2 – Importance of Computation

Very few engineering problems are simple enough to reasonably do a manual parametric sweep test.

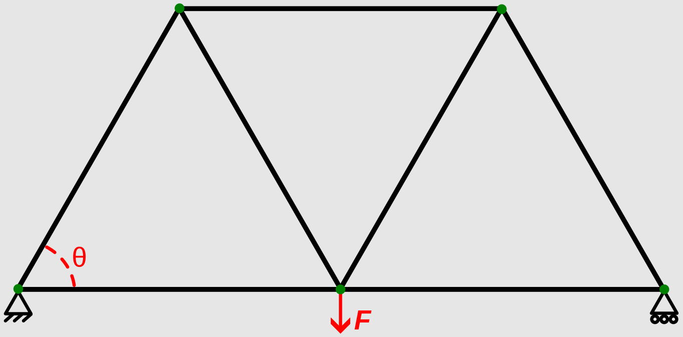
Even with the rover, you may not have time to test each material given that 40 other groups are also trying to make their parts. Not to mention architects and pesky bioengineers.

Computation empowers you to run thousands of tests before you ever approach the machines in our Makerspace.

3 – Warren Truss Problem Statement

✓ L.O.1
✓ L.O.2
➪ L.O.3

To illustrate this method, consider the simple Warren truss.



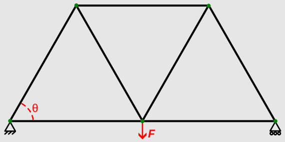
3 – Warren Truss Problem Statement

✓ L.O.1
✓ L.O.2
➩ L.O.3

To illustrate this method, consider the simple Warren truss.

This truss has two parameters: θ and L.

From the Statics 1 Project, $L = 18\,[\mathrm{in}]$. So, only θ is a parameter.



Our optimization metric is called the "Performance Index" (PI).

$$\mathrm{PI} = {}^{W_{\mathrm{held}}}\!/_{W_{\mathrm{truss}} C_{\mathrm{truss}}}$$

3 – General Points & Disclaimers

To perform this analysis, we'll vary θ and evaluate the bridge's performance for each iteration.

A few important notes before we dive in:

- I make simplifications here from the Statics 1 project;
- This analysis entirely neglects complexities like bending and distributed loading.
- This lecture will not cover the static system of equations;

3 – Point out Appendix

Because this lecture focuses on implementing optimization, and not Statics, the next few slides omit much detail.

The code for evaluating the bridge's performance will be included, but only in the Appendix.

We'll leave it as a black-box magic function until then.

You can reference the Appendix for all of those details!

3 – Bridge: Complete Main Code

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
rhs = @(node_load) [0;0;0;0;0;node_load;0;0;0;0];
theta_values = 1:89;     % test each angle from 1 -> 90
loads = zeros(1,89); PIs = zeros(1,89); previous_load = 0;
for theta = theta_values
  % <------ Method of Joints symbolic equations omitted for space ----->
  eqns = [Ax Ay Bx By Cx Cy Dx Dy Ex Ey];
  coeffs = double(equationsToMatrix(eqns));
  [load_capacity, performance] = evaluateBridge(previous_load,theta,...
                                          2886,1755,coeffs(:,1:end-1),rhs);
  previous_load = load_capacity;
  loads(theta) = 2*load_capacity; PIs(theta) = performance;
end
```

3 – Bridge: System Symbolic Variables

✓ L.O.1
✓ L.O.2
⇨ L.O.3

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
```

-
-
-
-
-
-
-
-
-
-
-
-

3 – Bridge: Warren Truss `rhs` Vector

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
rhs = @(node_load) [0;0;0;0;0;node_load;0;0;0;0];
.
.
.
.
.
.
.
.
.
.
.
```

3 – Bridge: Initialize θ Values to Test

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
rhs = @(node_load) [0;0;0;0;0;node_load;0;0;0;0];
theta_values = 1:89;    % test each angle from 1 -> 89
.
.
.
.
.
.
.
.
.
.
```

3 – Bridge: Initialize Looping Variables

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
rhs = @(node_load) [0;0;0;0;0;node_load;0;0;0;0];
theta_values = 1:89;
loads = zeros(1,89); PIs = zeros(1,89); previous_load = 0;
.
.
.
.
.
.
.
.
.
```

3 – Bridge: Set up `for` loop

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
rhs = @(node_load) [0;0;0;0;0;node_load;0;0;0;0];
theta_values = 1:89;
loads = zeros(1,89); PIs = zeros(1,89); previous_load = 0;
for theta = theta_values
    .
    .
    .
    .
    .
    .
    .
end
```

3 – Bridge: System of Equations

✓ L.O.1
✓ L.O.2
⇨ L.O.3

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
rhs = @(node_load) [0;0;0;0;0;node_load;0;0;0;0];
theta_values = 1:89;
loads = zeros(1,89); PIs = zeros(1,89); previous_load = 0;
for theta = theta_values
  % <----- Method of Joints symbolic equations omitted for space ----->
  eqns = [Ax Ay Bx By Cx Cy Dx Dy Ex Ey];
  coeffs = double(equationsToMatrix(eqns));
  .
  .
  .
  .
end
```

3 – Bridge: Evaluate Bridge

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
rhs = @(node_load) [0;0;0;0;0;node_load;0;0;0;0];
theta_values = 1:89;
loads = zeros(1,89); PIs = zeros(1,89); previous_load = 0;
for theta = theta_values
  % <----- Method of Joints symbolic equations omitted for space ----->
  eqns = [Ax Ay Bx By Cx Cy Dx Dy Ex Ey];
  coeffs = double(equationsToMatrix(eqns));
  [load_capacity, performance] = evaluateBridge(previous_load,theta,...
                                       2886,1755,coeffs(:,1:end-1),rhs);
  .
  .
end
```

3 – Bridge: Complete Main Code

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
rhs = @(node_load) [0;0;0;0;0;node_load;0;0;0;0];
theta_values = 1:89;
loads = zeros(1,89); PIs = zeros(1,89); previous_load = 0;
for theta = theta_values
  % <------ Method of Joints symbolic equations omitted for space ----->
  eqns = [Ax Ay Bx By Cx Cy Dx Dy Ex Ey];
  coeffs = double(equationsToMatrix(eqns));
  [load_capacity, performance] = evaluateBridge(previous_load,theta,...
                                      2886,1755,coeffs(:,1:end-1),rhs);
  previous_load = load_capacity;
  loads(theta) = 2*load_capacity; PIs(theta) = performance;
end
```

3 – Bridge: Plotting

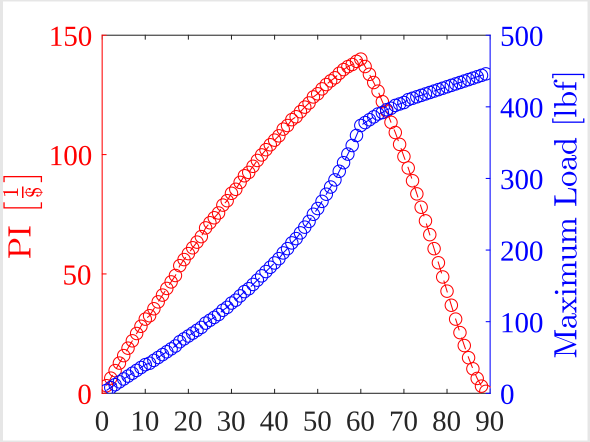
```
colororder({'r','b'})
yyaxis left
plot(theta_values, PIs, 'o--r')
ylabel('PI $\left[\frac{1}{\$}\right]$','Interpreter','latex');
yyaxis right
plot(theta_values, loads, 'o--b')
ylabel('Maximum Load $\left[\mathrm{lbf}\right]$','Interpreter','latex');
xlim([1,89]); xticks(0:10:90);
set(gca,'FontSize', 14, 'FontName', 'Times New Roman')
set(gcf,'PaperUnits','centimeters','PaperSize',[10 7.5], ...
        'PaperPosition',[0 0 10 7.5])
print('PI_Max-Load', '-dpdf')
```

3 – Optimization Results

This process produces the following figure:

There is a clear maximum PI right at $\theta = 60^\circ$.

That is the optimal θ for the Warren truss!



4 – Summary

This lecture covered:

✓ What optimization means in engineering

To maximize or minimize a performance metric for a system, especially under some constraint.

✓ What it means to optimize by parametric sweep

To evaluate the system for many values of one or more parameter. The best configuration is the optimized design.

4 – Summary

✓ Applying parametric sweep optimization to a bridge

In the context of the Statics 1 project, the variable θ is the only parameter. So optimization consists of varying that angle over the possible domain. The project defines a performance metric for us, PI, which we want to maximize. The sweep gives us a picture of the whole domain's performance.

5 – Navigating the Appendix

✓ L.O.1
✓ L.O.2
✓ L.O.3

The following slides provide all of the code used in this analysis (except for the static system of equations).

Each slide shows a body of code and is then followed with a slide that calls out any called functions within that body.

Those functions are then defined in the following slides. And so on until the end.

5 – Appendix: Main Code

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
syms RAx RAy REy AB AC BC BD CD CE DE applied_load
rhs = @(node_load) [0;0;0;0;0;node_load;0;0;0;0];
theta_values = 1:89;
loads = zeros(1,89); PIs = zeros(1,89); previous_load = 0;
for theta = theta_values
  % <----- Method of Joints symbolic equations omitted for space ----->
  eqns = [Ax Ay Bx By Cx Cy Dx Dy Ex Ey];
  coeffs = double(equationsToMatrix(eqns));
  [load_capacity, performance] = evaluateBridge(previous_load,theta,...
                                        2886,1755,coeffs(:,1:end-1),rhs);
  previous_load = load_capacity;
  loads(theta) = 2*load_capacity; PIs(theta) = performance;
end
```

5 – Appendix: evaluateBridge

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function [load_capacity, performance] = evaluateBridge(prev_load,...
                                         theta,t_yield,c_yield,coeffs,rhs_func)
  load = prev_load; maximum_t_stress = 0; maximum_c_stress = 0;
  while maximum_t_stress < t_yield && maximum_c_stress < c_yield
    load = load + 1;
    rhs = rhs_func(load);
    forces = coeffs \ rhs;
    [t_stress, c_stress, b_stress] = evaluateStresses(forces);
    maximum_c_stress = max([b_stress; c_stress]);
    maximum_t_stress = max(abs(t_stress));
  end
  load_capacity = load; performance = evaluatePerformance(load, theta);
end
```

5 – Appendix: evaluateBridge

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function [load_capacity, performance] = evaluateBridge(prev_load,...
                                      theta,t_yield,c_yield,coeffs,rhs_func)
  load = prev_load; maximum_t_stress = 0; maximum_c_stress = 0;
  while maximum_t_stress < t_yield && maximum_c_stress < c_yield
    load = load + 1;
    rhs = rhs_func(load);
    forces = coeffs \ rhs;
    [t_stress, c_stress, b_stress] = evaluateStresses(forces);
    maximum_c_stress = max([b_stress; c_stress]);
    maximum_t_stress = max(abs(t_stress));
  end
  load_capacity = load; performance = evaluatePerformance(load, theta);
end
```

5 – Appendix: evaluateBridge

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function [load_capacity, performance] = evaluateBridge(prev_load,...
                                        theta,t_yield,c_yield,coeffs,rhs_func)
  load = prev_load; maximum_t_stress = 0; maximum_c_stress = 0;
  while maximum_t_stress < t_yield && maximum_c_stress < c_yield
    load = load + 1;
    rhs = rhs_func(load);
    forces = coeffs \ rhs;
    [t_stress, c_stress, b_stress] = evaluateStresses(forces);
    maximum_c_stress = max([b_stress; c_stress]);
    maximum_t_stress = max(abs(t_stress));
  end
  load_capacity = load; performance = evaluatePerformance(load, theta);
end
```

5 – Appendix: evaluateStresses

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function [t_stress, c_stress, b_stress] = evaluateStresses(forces)
  in_tension = forces < 0; % method of joints convention
  tensile_forces = forces(in_tension);
  compressive_forces = forces(~in_tension);
  [t_area, c_area, b_area] = evaluateAreas(3/8, 3/8, 0.164);
  t_stress = tensile_forces ./ t_area;        % tensile
  c_stress = compressive_forces ./ c_area;    % compressive
  b_stress = abs(forces ./ b_area);           % bearing
end
```

5 – Appendix: evaluateStresses

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function [t_stress, c_stress, b_stress] = evaluateStresses(forces)
  in_tension = forces < 0; % method of joints convention
  tensile_forces = forces(in_tension);
  compressive_forces = forces(~in_tension);
  [t_area, c_area, b_area] = evaluateAreas(3/8, 3/8, 0.164);
  t_stress = tensile_forces ./ t_area;         % tensile
  c_stress = compressive_forces ./ c_area;     % compressive
  b_stress = abs(forces ./ b_area);            % bearing
end
```

5 – Appendix: evaluateAreas

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function [t_area, c_area, b_area] = evaluateAreas(xs_x, xs_y, d_bolt)
  t_area = (xs_x - d_bolt) * xs_y;
  c_area = xs_x * xs_y;
  b_area = xs_y * d_bolt;
end
```

5 – Appendix: `evaluatePerformance`

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function performance = evaluatePerformance(load, theta)
    performance = 2*load / (2*evaluateCost(theta, 5) * ...
                            2*evaluateWeight(theta, 5));
end
```

5 – Appendix: evaluatePerformance

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function performance = evaluatePerformance(load, theta)
    performance = 2*load / (2*evaluateCost(theta, 5) * ...
                            2*evaluateWeight(theta, 5));
end
```

5 – Appendix: `evaluatePerformance`

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function performance = evaluatePerformance(load, theta)
    performance = 2*load / (2*evaluateCost(theta, 5) * ...
                            2*evaluateWeight(theta, 5));
end
```

5 – Appendix: evaluateCost

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function cost = evaluateCost(theta, num_nodes)
  balsa_cost_per_inch = 0.0407;    % Balsa wood cost     [$/in]
  washer_cost_per_unit = 0.0411;   % Washer cost         [$/unit]
  nut_cost_per_unit = 0.0457;      % Nut cost            [$/unit]
  bolt_cost_per_unit = 0.1564;     % 1.5" Bolt cost      [$/unit]
  [~, ~, L_tot] = evaluateLength(theta);
  cost = sum([washer_cost_per_unit*(2*num_nodes), ...
              nut_cost_per_unit*(num_nodes), ...
              bolt_cost_per_unit*(num_nodes), ...
              balsa_cost_per_inch*(L_tot)]);
end
```

5 – Appendix: evaluateWeight

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function weight = evaluateWeight(theta, num_nodes)
  hardware_weight_per_node = 0.01;  % Weight of all hardware per node
  balsa_density = 0.0013;           % define the density [lbf/in]
  [~, ~, L_tot] = evaluateLength(theta);
  weight = balsa_density * L_tot + ...
           hardware_weight_per_node * num_nodes;
end
```

5 – Appendix: evaluateWeight

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function weight = evaluateWeight(theta, num_nodes)
  hardware_weight_per_node = 0.01;  % Weight of all hardware per node
  balsa_density = 0.0013;           % define the density [lbf/in]
  [~, ~, L_tot] = evaluateLength(theta);
  weight = balsa_density * L_tot + ...
           hardware_weight_per_node * num_nodes;
end
```

5 – Appendix: evaluateLength

✓ L.O.1
✓ L.O.2
✓ L.O.3

```
function [L_truss, L_main, L_tot] = evaluateLength(theta)
  l = 9;
  h = tand(theta);
  A = [0,0];
  B = l*[0.5,h];
  C = l*[1,0];
  D = l*[1.5,h];
  E = l*[2,0];
  L_truss = norm(B-A) + norm(C-B) + norm(D-B) + norm(E-D);
  L_main = 18;
  L_tot = L_truss + L_main;
end
```